

EECS 349 (Machine Learning) Homework 4

WHAT TO HAND IN

You are to submit the following things for this homework:

1. A **PDF** document containing answers to the homework questions.
2. The **WELL COMMENTED** source code for all software you write.

HOW TO HAND IT IN

To submit your assignment:

1. Compress all of the files specified into a .zip file.
2. Name the file `firstname_lastname_hw4.zip`. For example, `Bryan_Pardo_hw4.zip`.
3. Submit this .zip file via Canvas.

1) Hopfield Net (3 points + 1 bonus point)

A. (1/2 point) Implement a Hopfield Network training method/function that conforms to the following specification.

```
def train_hopfield(X):
    %train_hopfield(X)
    %
    % Creates a set of weights between nodes in a Hopfield network whose
    % size is based on the length of the rows in the input data X.
    %
    % X is a numpy.array of shape (R, C). Values in X are drawn from
    % {+1,-1}. Each row is a single training example. So X(3,:) would be
    % the 3rd training example.
    %
    % W is a C by C numpy array of weights, where W(a,b) is the connection
    % weight between nodes a and b in the Hopfield net after training.
    % Here, C = number of nodes in the net = number of columns in X
    %
    % return W
```

B. (1/2 point) Implement the following method to use a Hopfield network, once it is trained.

```
def use_hopfield(W,x):
    %use_hopfield(W, x)
    %
    % Takes a Hopfield net W and an input vector x and runs the
    % Hopfield network until it converges.
    %
    % x, the input vector, is a numpy vector of length C (where C is
    % the number of nodes in the net). This is a set of
    % activation values drawn from the set {+1, -1}
    %
    % W is a C by C numpy array of weights, where W(a,b) is the connection
    % weight between nodes a and b in the Hopfield net.
    % Here, C = number of nodes in the net.
    %
    % s is a numpy vector of length C (number of nodes in the net)
    % containing the final activation of the net. It is the result of
    % giving x to the net as input and then running until convergence.
    %
    % return s
```

EECS 349 (Machine Learning) Homework 4

C. (1 point) The Semeion handwriting digit data set is available here:

<https://archive.ics.uci.edu/ml/datasets/Semeion+Handwritten+Digit>

(NOTE: You will need to change how the data is encoded for it to work with a Hopfield net, since a Hopfield net expects -1 and 1 as the input).

Select a set of single examples of the digits $\{0, 1, 2, 3, 4\}$ from a single writer. Train a Hopfield network on this set of digits. Now, select a second example of one of these digits. **Pick a 2nd example of JUST ONE of these digits** (one that the network was not trained on). Pass it as an input to the trained network (i.e. use `_hopfield`). Show the progression of the state of the network for 10 steps (where 1 step = 1 update of ALL nodes in the net), or convergence, whichever comes first. Did it converge to the right digit? Do you think it is close enough to build a classifier with it?

(BONUS EXTRA CREDIT QUESTION) D. (1 point) Train the Hopfield net on just two digits (e.g. 1 and 3). Take an example of one of the digits it was actually trained on. Randomly flip 20% of the 256 bits to create a noisy input. Pass this as input to the trained network. Try it a few times with different digit pairs to see what happens. Did it always converge to the right digit? If not, what did happen? For one of your digit pairs, show the progression of the state of the network for 10 steps (where 1 step = 1 update of ALL nodes in the net), or convergence, whichever comes first. Does the series look similar to the one shown in the lecture slides drawn from the article An Introduction to Computing with Neural Nets, IEEE ASSP Magazine, April 1987?

E. (1 point) What happens to the functioning of a Hopfield network if you replace inputs drawn from $\{+1, -1\}$ with inputs drawn from $\{0, 1\}$? Explain your answer.

2) About Expectation Maximization (4 points)

A. (1 point) If you have a single (univariate) Gaussian, you can maximize the expected likelihood of observing the data by taking partial derivatives with respect to the parameters of the Gaussian. Show that this is true (i.e. do the math).

B. (1 point) There isn't a single closed-form solution to finding the mixture model that maximizes the likelihood of the data when the number of Gaussians in your GMM (Gaussian Mixture Model) exceeds 1. Give an explanation of why.

C. (1 point) Since there isn't a closed form solution to finding the maximum likelihood model parameters, we need another approach. Expectation Maximization (EM) is the standard method for tuning model parameters. Sections 8.5.2 and 8.5.3 in Elements of Statistical Learning go through two views of why Expectation Maximization algorithm gives monotonically non-decreasing probability of generating the data as the model parameters are updated. Pick one. Paraphrase it in your own words. Use some math. Define all your variables.

D. (1 point) What if there were a data point at the exact value of μ for some Gaussian in our GMM. What would the update rules in the course notes cause to happen? Is this good or bad? If good, say why. If bad, say why and give a fix that would prevent this.

EECS 349 (Machine Learning) Homework 4

3) Making Gaussian Mixture Models (4 points)

Load “gmm_test.csv” and “gmm_train.csv”. This contains two labeled samples of data: X_{test} and X_{train} . The labels for each sample are contained in Y_{test} and Y_{train} . So $Y_{\text{train}}(i)$ contains the label for $X_{\text{train}}(i)$. Each of these samples was drawn from data coming from two different underlying populations, labeled as classes “1” and “2”.

In this problem, you will model each class in the training data, using a GMM for each class. You will then design a classifier to classify X_{test} using your GMMs. You will evaluate your classification results on the test data. You need to submit your source code of “gmmost.py” and “gmmlclassify.py” as described below.

Note: You are welcome to make your functions work for multivariate (a.k.a. multidimensional) GMMs, but you are not required to do so.

A. (1 point) Implement the EM algorithm to estimate parameters (means, variances, weights) of a 1-dimensional GMM. Name your function “gmmost.py”.

The function should be called like this:

```
def gmmost(X, mu_init, sigmasq_init, wt_init, its):
    % Use numpy vectors/arrays.
    % Input
    %   - X           : N 1-dimensional data points (a 1-by-N vector)
    %   - mu_init     : initial means of K Gaussian components
    %                   (a 1-by-K vector)
    %   - sigmasq_init: initial variances of K Gaussian components
    %                   (a 1-by-K vector)
    %   - wt_init     : initial weights of k Gaussian components
    %                   (a 1-by-K vector that sums to 1)
    %   - its         : number of iterations for the EM algorithm
    % Output
    %   - mu          : means of Gaussian components (a 1-by-K vector)
    %   - sigmasq     : variances of Gaussian components (a 1-by-K vector)
    %   - wt          : weights of Gaussian components (a 1-by-K vector, sums
    %                   to 1)
    %   - L           : log likelihood
```

B. (1point) Test your function by building a mixture model for each of the two classes in X_{train} . Choose an appropriate number of Gaussian components and initializations of parameters. Plot the data log-likelihood values for the first 20 iterations. Do you think your program has converged? Report the final values of the GMM parameters for class 1 and for class 2.

Hint: To pick good initializations, visualize the histogram of each class of data. The 'hist' function in matplotlib is a big help. So is the 'nonzero' function in numpy. Try this example code to get started. (Y_{test} and X_{test} should be numpy arrays)

```
import numpy as np
import matplotlib.pyplot as plt

class1 = X_test[np.nonzero(Y_test ==1)[0]]
class2 = X_test[np.nonzero(Y_test ==2)[0]]
bins = 50 # the number 50 is just an example.
plt.subplot(2,1,1)
plt.hist(class1, bins)
plt.subplot(2,1,2)
plt.hist(class2, bins)
plt.show()
```

EECS 349 (Machine Learning) Homework 4



C. (1 point) Implement a function to perform binary classification using your learned GMMs. Name your function “gmmclassify.py”.

The function should be called as

```
def gmmclassify(X, mu1, sigmasq1, wt1, mu2, sigmasq2, wt2, p1):  
    % Use numpy vectors/arrays.  
    % Input  
    % - X          : N 1-dimensional data points (a 1-by-N vector)  
    % - mu1         : means of Gaussian components of the 1st class  
    %                 (a 1-by-K1 vector)  
    % - sigmasq1    : variances of Gaussian components of the 1st class  
    %                 (a 1-by-K1 vector)  
    % - wt1         : weights of Gaussian components of the 1st class  
    %                 (a 1-by-K1 vector, sums to 1)  
    % - mu2         : means of Gaussian components of the 2nd class  
    %                 (a 1-by-K2 vector)  
    % - sigmasq2    : variances of Gaussian components of the 2nd class  
    %                 (a 1-by-K2 vector)  
    % - wt2         : weights of Gaussian components of the 2nd class  
    %                 (a 1-by-K2 vector, sums to 1)  
    % - p1          : the prior probability of class 1.
```

D. (1 point) Run gmmclassify on X_{test} with the two GMMs your learned and the class prior probability p_1 calculated from the training data. Report on the accuracy of your classifier (Just a single number, no need to do n-fold validation or statistics on this problem). Make a two-color histogram of the two classes in X_{test} , where the color indicates the ground-truth correct class for the data (see the example code in the hint above for a starting point). Then show classes learned by your GMM on this same plot by putting a colored marker (where the color indicates the class chosen by your classifier) on the horizontal axis for each classified data point.

4) Reinforcement Learning (4 points)

MINI Wumpus World			
A	r = 0	r = 0	r = 50 
	r = 0	r = -10 	r = 0
B			
C	r = 0	r = 0	r = 0
	1	2	3

Adventure 1 = {C1, B1, B2}

Adventure 2 = {C1, B1, A1, A2, A3}

Adventure 3 = {C1, C2, B2}

Adventure 4 = {C1, C2, C3, B3, A3}

Adventure 5 = {C1, C2, C3, B3, B2}

The mini Wumpus world is shown above. In this world, you always start in state C1. Thereafter, you may move horizontally or vertically (not diagonally) to any adjacent state. Thus, the set of moves are U,D,L,R (standing for up, down, left and right). Moves outside the world are impossible. For a state such as C1, there are only 2 possible moves: U and R. An adventure ends when you either encounter the Wumpus (a little red dragon) or find the pot of gold. Here, the

EECS 349 (Machine Learning) Homework 4

immediate reward for being in any one state is shown to you. You also have the history of locations recorded for our hero on five adventures in the mini Wumpus world.

A. (2 points) Given the mini Wumpus world and the five adventures (shown above), approximate the Q-function for each allowed state-action pair and show the Q-function in a table. The algorithm for this is in Table 13.1 of the textbook (Mitchell). Show your work. If you wrote a computer script to do the work, submit the source code in a zip file with the answers to these questions. Code should be commented and a readme should explain how to use it. If you did it by hand, show the steps. Assume your discount factor $\gamma = 0.5$ and process the adventures in ascending order: Adventure 1, then Adventure 2 and so on.

B. (2 point) Give the value of the policy function for each state, given the Q-function approximation you learned in part A.